1 Kjøretid Sliding

Se på denne java koden og analyser kjøretiden som funksjon av `n`. Du skal velge det svaret som best beskriver kjøretiden.

```
public static ArrayList<Double> slidingWindow(ArrayList<Double> data) {
   ArrayList<Double> filtered = new ArrayList<Double>();
   int n = data.size();

   for(int i = 0; i < n; i ++) {
       int low = Math.max(0, i - 5);
       int hi = Math.min(n - 1, i + 5);
       double sum = 0.0;

       for (int j = low; j <= hi; j++) {
           sum += data.get(j);
       }

       double avg = sum (hi - low + 1);
       filtered.add(avg);
   }
   return filtered;
}
```

Svar. The runtime will be $O\left(n\sqrt{n}\right)$, because the inner for loop will iterate on average $\sqrt{n}$ times.

2 Kjøretid reverse LinkedList

Se på denne java koden og analyser kjøretiden som funksjon av `n` = lengden på `list`. Du skal velge det svaret som best beskriver kjøretiden.

```
public static <T> LinkedList<T> reverse (LinkedList<T> list) {
   LinkedList<T> reverse = new LinkedList<T>();
   for(T t : list) {
       reverse.addFirst(t);
   }
   return reverse;
}
```

Svar. Metoden `addFirst` kjører i konstant tid, og derfor har denne metoden kjøretid $O(n)$.

3 Kjøretid moveFirst

Se på denne java koden og analyser kjøretiden som funksjon av `n` = lengden på `list` og `k` = lengda på `toMove`. Du skal velge det svaret som best beskriver kjøretiden.

```
public static <T> void moveFirst(ArrayList<T> list, ArrayList<T> toMove) {
   int n = list.size()
```

```
    for(int i = 0; i < n; i++) {
        T elem = list.get(i);
        if (toMove.contains(elem)) {
            list.remove(i);
            list.add(0, elem);
        }
    }
}
```

Svar. For løkken itererer `n` ganger, og det sjekkes hver gang om et element er i listen `toMove` med `contains()` metoden. Denne metoden har kjøretiden $O(k)$. Videre har metodene `remove()` og `add(0, elem)` worst-case kjøretider på $O(n)$. Worst-case for `if` setningen vil derfor være $O(n+k)$. Vi får da en endelig kjøretid på $O(n(n+k))$.

4 Speedy

Du har inngått et veddemål med din venn "Speedy", han mener han klarer å løpe fra hybelen til forelesningssalen på mindre enn en halv time mens du tror han vil bruke mer.

For å øke sjansene dine har du bestemt deg for å be din venn som jobber i kommunen om å blokkere 1 vei slik at Speedy vil bruke litt lenger tid.

Du har modellert veinettet som en vektet graf med n noder og O(n) kanter der hver kant er vektet med tiden Speedy vil bruke på å løpe den strekningen som kanten representerer.

Speedy vil få vite hvilken vei som er blokkert før han begynner å løpe og løper den korteste veien til universitetet.

Beskriv en effektiv algoritme som bestemmer hvilken gate/kant du bør blokkere for at Speedy skal bruke lengst mulig tid.

Svar. Her kan vi lage en algoritme som først fjerner en kant i grafen, for å deretter utføre en korteste-sti på grafen med den fjernede kanten. Korteste-sti algoritmen vi bruker kan være f.eks *Dijkstra's Algoritme*. Vi kan lagre kantene vi har fjernet, sammen med lengden til den korteste stien i grafen hvor denne kanten er fjernet. Deretter velger vi den kanten som gir *lengst* kortest sti.